Movies, Let's See Some Movies

Zhizhong Li
Dec 12, 2016
Information Engineering, CUHK
1155070507, lz015@ie.cuhk.edu.hk

1 Introduction

In this report, we deal with the low-rank approximation problem. Given an $n \times d$ matrix A, we would like to find a matrix $\tilde{A}_k$, such that

$$\|A - \tilde{A}_k\| \leq (1 + \varepsilon)\|A - A_k\|, \tag{1}$$

where A_k is the best rank-k approximation oto A, and $\tilde{A}_k$ has rank k. The best approximation A_k can be computed by a svd operation. However, the time required would be huge. We tested several approximation methods using the sketching technique, both in the Frobenius norm and the operator norm settings.

1.1 Dataset

We use the MovieLens 20M dataset [2] as benchmark, which can be downloaded at [1]. It contains 20,000,263 5-star ratings on 26744 movies from 138493 users. This is a typical dataset for the study of recommendation systems. The low rank nature for the problem is that the contributing factors for users' preference on a movie is not that much. Factors like genre, director, actors, etc might dominate one's selection. The data is also sparse since the number of movies that a user can reach is limited. We put all ratings in a sparse matrix

$$A \in \mathbb{R}^{138493 \times 26744}, \tag{2}$$

with columns be different movies and rows represent users. There are 20,000,263 (5.4%) none-zero entries, and all elements range in $[0, 5]$.

1.2 Experiment Setting

All experiments were conducted on a machine that contains an 80-core Intel Xeon E5-2698 v4 CPU at 2.2GHz, and 512G memory. Codes were implemented using the Julia language and are available at https://github.com/innerlee/random.computation.report1.

2 Baseline

We did a svd directly to the full version (*i.e.*, none sparse algorithms used) of the matrix, which took

$$t_0 = 21403 \tag{3}$$

seconds, about 6 hours. The eigenvalues was shown in Figure 1. We can see that the dominating values only take a small proportion. For example, known that the largest eigenvalue is 7383.58, there are only 81 eigenvalues that are larger than 500, and when we set the threshold to be 1000, merely 15 of them survive. So it is reasonable to use a low-rank matrix to approximate the data. We also show the distribution of eigenvalues that are larger than 500 in Figure 1.

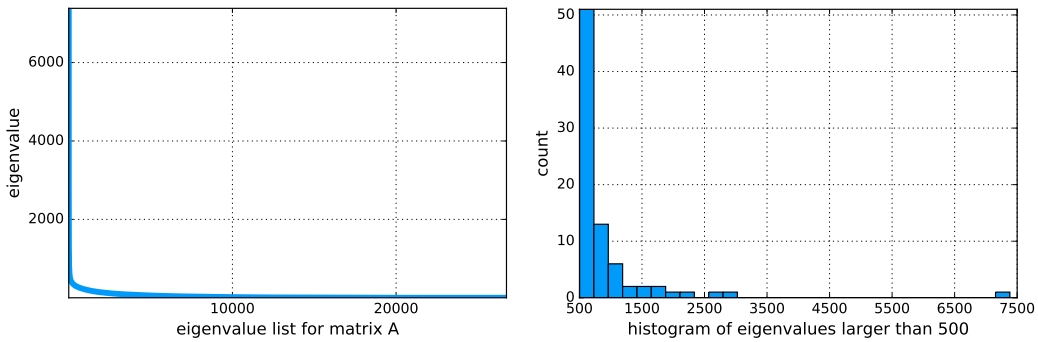

Figure 1: **Left**: Eigenvalues of matrix A. They are sorted for visualization. We can see a small portion of very large values. **Right**: Detailed distribution for large eigenvalues.

To find a gook rank k, we plot the curve of error energy (squared Frobenius norm) v.s. rank k in Figure 2. Since there are quick methods (like sparse svd function *svds()* in Julia) computing part of the svd given rank k, The running time of svds() is shown in Figure 2. We can see that when $k = 128$, the relative error energy is around 50% and the running time for svds in this setting is

$$t_1 = 111.4 \tag{4}$$

seconds. We will set

$$k = 128 \tag{5}$$

in the following experiments since it captures half of the energy as well as keeps a relative small rank at the same time. The Frobenius norm

$$\|A - A_{128}\| = 11622.2. \tag{6}$$

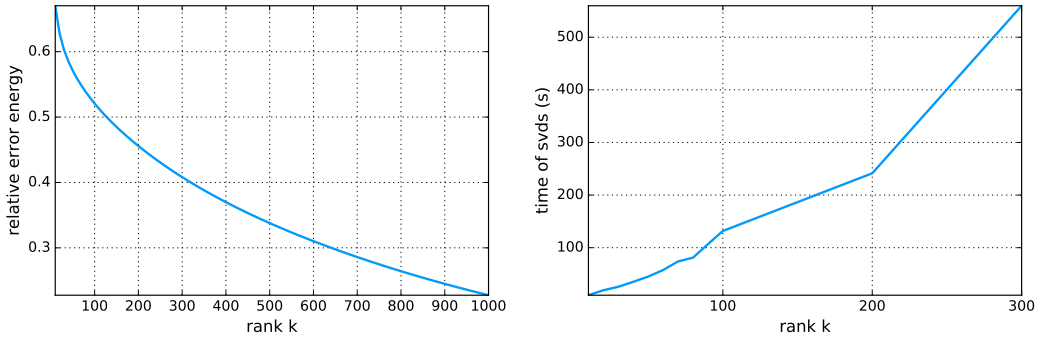

Figure 2: **Left**: Relative errors of the best rank k approximation to A which are measured in the Frobenius norm squared. **Right**: The running time of sparse svd, svds(), in different k.

3 Sketching

In this section, we implemented the low-rank approximation of matrix A in two settings: Frobenius norm and operator norm. Most of the algorithm is described in the textbook article [3].

3.1 Frobenius norm

For simplicity, we use sparse embedding, *i.e.* the count sketch to find a subspace that may contain a good low-rank approximation to A. The general steps in *sketch_frob()* function are:

1. Input: matrix $A \in \mathbb{R}^{138493 \times 26744}$, rank $k = 128$, $t \in \{256, 512, 1024\}$ which is the the sketch dimension.
2. Construct count sketch matrix S.

3. Do qr decomposition $Q, R = \text{qr}((SA)^T)$.
4. Do svd $U, S, V = \text{svd}(AQ)$.
5. Truncate to k-dimensions U_k, S_k, and V_k.
6. Output: $\tilde{A}_k = U_k S_k V_k^T Q^T$.

We divide the process into two phases and count the running time independently. The first phase is sketching and the second phase includes all the remaining steps which is to solve a reduced version of the problem. Results are shown in Table 1.

Table 1: The performances of different sketching techniques. k is the rank used, t is the dimension for sketching. time-i is the average time spent on phase i. Errors are measured by Frobenius norms.

algorithm	repeat	k	t	time1&2	time (s)	time ratio	min err	max err	mean err	std err	err ratio
Baseline	1	128	-	-	111.4	1	11622.2	11622.2	11622.2	-	1
Sketching	100	128	256	2.7/13.8	16.5	0.148	12664.1	12693.7	12677.0	6.2537	1.089
Sketching	100	128	512	2.8/23.8	26.6	0.239	12282.5	12298.3	12290.0	3.1376	1.056
Sketching	100	128	1024	2.8/49.8	52.6	0.472	11971.6	11979.0	11974.7	1.314	1.030

3.2 Operator norm

The operator norm problem was tackled by the subspace power method. We implemented it in the *sub_power()* function. The general steps go like this,

1. Input: matrix $A \in \mathbb{R}^{138493 \times 26744}$, rank $k = 128$, power $q \in \{10, 20, 40\}$.
2. Generate random Gaussian matrix $G \in \mathbb{R}^{26744 \times k}$.
3. $Y = AG$.
4. repeat $Y = A^T Y, Y = AY$, for q times.
5. Do qr factorization $Z, \_ = \text{qr}(Y)$.
6. Output: $\tilde{A}_k = ZZ^T A$.

The results are shown in Table 2. Since the running time and evaluation time for each run is too slow, we reduced the repeat number to 16.

Table 2: The performances of subspace power method. k is the rank used, q is the power used. time is the in seconds and errors are measured by operator norms.

algorithm	repeat	k	q	time (s)	time ratio	min err	max err	mean err	std err	err ratio
Baseline	1	128	-	111.4	1	432.6	432.6	432.6	-	1
Sub Power	16	128	4	53.3	0.479	460.0	470.9	466.6	3.031	1.063
Sub Power	16	128	8	88.4	0.794	663.3	694.6	683.6	8.984	1.533
Sub Power	16	128	16	165.1	1.482	1908.6	1960.7	1933.3	15.113	4.412

The result is astonishing. By doing powers more, we get worse performance. What is the problem then? We analyzed the matrices in the middle of computations and found two problems,

1. Without renormalize, elements in matrix Y soon explode to extremely large values. This may lead to numerical issues in the later qr computation.
2. Simply scale the matrix to a lower level would not effect that well as expected. We then found that column vectors in Y did not orthogonal with each other. In fact, we computed the some of the pair-wise angles and found that they are quite close to collinear after many power iterations.

By the above analysis, we made modifications to the algorithm by adding an extra qr step in the loop. *i.e.*, from

- repeat $Y = A^T Y,\ Y = AY$, for q times.

to

- repeat $Y = A^T Y,\ Y = AY,\ Y, \_ = \text{qr}(Y)$, for q times.

New results are listed in Table 3.

Table 3: The performances of the fixed version of subspace power method, in which an extra qr step was added to the power loop. k is the rank used, q is the power used. time is the in seconds and errors are measured by operator norms.

algorithm	repeat	k	q	time (s)	time ratio	min err	max err	mean err	std err	err ratio
Baseline	1	128	-	111.4	1	432.6	432.6	432.6	-	1
Sub Power (fix)	16	128	4	53.3	0.479	460.0	470.9	466.6	3.031	1.063
Sub Power (fix)	16	128	8	99.2	0.890	445.5	454.7	450.7	2.273	1.029
Sub Power (fix)	16	128	16	178.2	1.600	437.9	443.9	440.9	1.820	1.012

The extra qr step added computation time to the algorithm. Nevertheless, the accuracy also grows.

4 Discussion

From the experiments, we can observe that

- In the Frobenius norm setting, most of the time was spent on the second phase, *i.e.* solving the smaller sized problem. Specifically, Almost all time was used in the multiplication AQ.
- To accelerate this multiplication, we tried multiple methods. The interesting fact is that, by converting matrix A from sparse to full, the multiplication will be much faster (50s v.s. 5s). This might due to cache or memory alignment or so. It would be good if sketching that keep operator norm of the product of two matrices can be used here.
- The error of sketching in different runs is stable as we can see from the small std values. The relative error is below 5% when sketching to $t = 1024$ dimension. Smaller t leads larger error. Due to to time consuming multiplication, the time gain is not ideal. In the $t = 1024$ case, we only achieved a 2x speed up.
- The original algorithm low-rank approximation under operator norm has some numerical issues as we discussed in the corresponding section. We solved the problem by inserting an additional normalize operation.
- If we want to get high precision using subspace power method in the operator norm case, the running time may be not favorable as we have seen in the last row of Table 3. This is because the method contains lots of matrix multiplications, which are time consuming.
- In both of the cases, sketching method for matrix multiplication that keep operator norm is wanted since the speed bottleneck lies in the multiplication operation.

References

[1] GroupLens. MovieLens 20M dataset, accessed Dec 2016. URL `http://grouplens.org/datasets/movielens/20m/`.

[2] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, Dec. 2015. ISSN 2160-6455.

[3] D. P. Woodruff. Sketching as a tool for numerical linear algebra. *arXiv preprint arXiv:1411.4357*, 2014.